WEBVTT

1

00:00:03.560 --> 00:00:08.750

Jisun An: Alright! I can. Oh, oh, why it's not!

2

00:00:11.060 --> 00:00:14.790

Jisun An: Oh, didn't know that they need

3

00:00:14.980 --> 00:00:29.609

Jisun An: was not turned on all right. So today's passcode is Reg. RAG. The retriever augmented generation, which is the topic of today. We have very low attendance today. What happens?

4

00:00:30.241 --> 00:00:33.190

Jisun An: It's interesting to see that like.

5

00:00:34.570 --> 00:00:58.710

Jisun An: I I think people are still coming. A few announcements. And if you have like team members, I think it would be great if you can relay the message if they are missing this. So I mean so up to last week, I think that was all, mostly about principles of the large language model. So all the basic components fundamental blocks have been now built

6

00:00:58.710 --> 00:01:05.952

Jisun An: and the rest of the semester will be mostly on like applications. And

7

00:01:07.196 --> 00:01:23.883

Jisun An: some some variants of the Rrms. But hopefully, these application potentially can help for your project. But I I guess the hard part has been done, and the rest I feel that is far less difficult than the

8

00:01:24.700 --> 00:01:39.930

Jisun An: than the previous ones that we've seen. So a few things are coming up. So, firstly, the theoretical assignment, 2 will be posted tonight. So, and we you will have about 10 days until the end of the next week.

9

00:01:40.285 --> 00:02:00.200

Jisun An: To submit this, and the format will be very similar to what you've seen in the 1st assignment. It will be multi-choice questions it will cover like from the pre-training to the rag which we'll cover today. Once again. These are for like reminding you of the concepts that we have dealt with, and

10

00:02:00.480 --> 00:02:05.579

Jisun An: to help you to clarify some of the ideas and the concepts that we've been talking about.

11

00:02:06.140 --> 00:02:14.679

Jisun An: And another thing is the project proposal presentation will take place next week. So here are a few

12

00:02:15.000 --> 00:02:19.460

Jisun An: thing. So each presentation should be 6 min.

13

00:02:19.820 --> 00:02:40.130

Jisun An: and you can either present it present it. Live lively in the classroom next week, or you can also send me a pre-recorded presentation as well. But and the slides or the recordings should be submitted by March 10, th which is the day before the Tuesday. So next Monday.

14

00:02:40.881 --> 00:03:04.719

Jisun An: and so the details of suggested slide structures and the evaluation. Rubrics are now posted on the canvas, so please look check them. But then given, this is the proposals, and I also assume that there may be a team that may change the proposal entirely as well, and I will be as generous as possible for this. But hopefully this will be a good

15

00:03:04.820 --> 00:03:13.159

Jisun An: opportunity for your team to kickstart the project so that we can spend the rest of the term to finish this up.

16

00:03:14.080 --> 00:03:34.020

Jisun An: So next Tuesday and Thursday will be the presentation schedule. Altogether we have 16 teams, and each day 8 teams will present, and we who will present on Tuesday will be randomly selected on March 11th next Tuesday, so I hope all the teams are joining or at least prepared

17

00:03:34.301 --> 00:03:50.639

Jisun An: but if you have any particular scheduling conflict. Let me know, then I will try to accommodate, as I mean, if you have preference or not preference, but hard constraint, on which day you cannot present. But let me know. Then I will avoid that day. And then we can select randomly based on those numbers.

18

00:03:51.760 --> 00:03:52.920

Jisun An: Oh.

19

00:03:53.560 --> 00:04:18.370

Jisun An: yeah, but I mean, you can check the canvas. But it's like starting with introduction motivation related work research questions or the goals preliminary data set, or the method or so basically like outlines of what? How you would design the entire projects some discussions and the feasibilities and the reference. So this will be the some of the components that would be requiring for the proposal.

20

00:04:18.690 --> 00:04:25.089

Jisun An: I mean it. It may sounds a lot, but it's it's a good way to structure, your presentation, so

21

00:04:25.200 --> 00:04:31.059

Jisun An: trying to fill each of the components, I think that will help you to understand better about your own projects as well.

22

00:04:31.993 --> 00:04:36.380

Jisun An: Yeah, let me know if you have any questions about the proposal or presentation.

23

00:04:37.210 --> 00:04:44.470

Jisun An: But once again I mean given that it's the 1st round of the projects. I I at the moment. I also don't have

24

00:04:45.000 --> 00:04:53.729

Jisun An: I? I don't know what to expect at the moment, so I will. I will really looking forward to seeing what what you're thinking. So yeah, hopefully, it'll be a fun time.

25

00:04:54.000 --> 00:05:05.480

Jisun An: And I think that's all. So for those who just joined today's passcode reg. Please mark your attendance.

26

00:05:07.320 --> 00:05:10.000

Jisun An: Any questions regarding the logistics?

27

00:05:13.950 --> 00:05:14.780

Jisun An: Okay?

28

00:05:15.820 --> 00:05:23.599

Jisun An: Right? So today's topic is the retriever augmented generation. And I think

29

00:05:25.250 --> 00:05:44.130

Jisun An: this is it's an easy topic. And also I I feel like this is not really about Edith, but more about information retriever. So the contents is more similar to Ir content. But but this has been a in the center of modern based systems, because

30

00:05:44.537 --> 00:05:56.360

Jisun An: just purely using Rnm, only has a few issues, and our Ag has been helped to solve those issues. So I think it became one of the

31

00:05:56.990 --> 00:06:00.650

Jisun An: central ingredients of the database systems.

32

00:06:00.750 --> 00:06:12.380

Jisun An: But at the same time I also have a bit of doubt whether how, until when our age will be really really needed, maybe for the company still needed. So we can talk more about that soon.

33

00:06:13.430 --> 00:06:38.410

Jisun An: So so why read is coming out, I mean introduced in the 1st place. So assuming, like standard prompting is so, we have some question like instruction like, please answer this question, and assuming that this was the question like, so I think when detail has been a voice actor for several characters in TV series. Do you know what their names are? So assuming that you ask this question

34

00:06:38.410 --> 00:06:44.899

Jisun An: to like Chat Gpt. Or many other anyone knows what character he has been

35

00:06:44.900 --> 00:06:47.919

Jisun An: would be playing as a voice actor.

36

00:06:48.260 --> 00:06:51.900

Jisun An: Oh, he is the guy from the fast and furious. The guy

37

00:06:52.720 --> 00:07:07.530

Jisun An: good. Oh, yeah. How do you know that? I I didn't know so anyway. So he is the person who did a voice actor for the group. I'm good and that was really funny so assuming that you had this question as a prompt to the

38

00:07:07.740 --> 00:07:18.848

Jisun An: to the Uhm's then, and assuming that you got some answers which will basically list a few like TV series that that this person has been

39

00:07:19.400 --> 00:07:20.930

Jisun An: participated.

40

00:07:21.150 --> 00:07:42.190

Jisun An: And then. So even though I'm not showing the actual features, and because the modern Rrms are so good at answering this. So if you are giving this prompt to like Chatgpt or any other closed models. I think they will be very good at answering to this question, but even just for, like 6 months ago, I think there was still this kind of problem that this

41

00:07:42.190 --> 00:08:02.509

Jisun An: to answer to this kind of problem, I mean problems. There could be 2 different problems. One would be accuracy issue. So basically, they will answer something, but they will, they may not be accurate. And the reason that why there could be an accuracy issue, there could be multiple regions personally, knowledge cut off. So

42

00:08:03.035 --> 00:08:20.324

Jisun An: because these are basically trained model. So if the data was trained, if the data used for training was collected until last month. Then, whatever happens this month, basically do not be part of the trained. So that could be one of the reason why the accuracy may not be

43

00:08:20.980 --> 00:08:41.120

Jisun An: high, or it could be also an issue of the private data. So if you are basically working for a particular company, and if you are working, I mean looking for those data, then these are usually not part of the this data set of these. So information about this private data will be not

44

00:08:41.730 --> 00:09:03.759

Jisun An: extracted directly. And also even there are some data there's there still be some issues with the learning it for different regions. Some information may not be learned correctly or neglected from during the training purposes, so these would be potential reason that the visuals might have some accuracy issue

45

00:09:03.760 --> 00:09:18.549

Jisun An: and also verifiability issue. Basically as a human because we touch it. I mean, Edms are so good at generating the the text that is look like like written, I mean, very like, well structured and coherent to text.

46

00:09:18.550 --> 00:09:27.568

Jisun An: And it it sounds like really like a good answer. It would be very hard to tell. Whether, if if the answer is correct or not, and this is something that

47

00:09:28.090 --> 00:09:45.519

Jisun An: oh, that's a hallucination problem. So rig is partly to solve this particular issue. So basically, when the Richards came out, can we really believe that Richards, and at the same time, can we improve the accuracy of that Richards.

48

00:09:45.954 --> 00:10:04.770

Jisun An: And especially so if you think about why this problem also happens, then there could be cases where our are not confident in answering something. Right? So for those cases, these 2 issues might be happening. And our Ag is there to support or tackle this problem.

49

00:10:05.570 --> 00:10:29.770

Jisun An: So I mean the the concept itself was there for a long time, but then, I think, in 2020 there was a 1st paper that used this term retriever augmented generation in particular, which I will introduce at the end of this lecture. But the flow of the rag is something like it. So the goal is given a query, we want to efficiently retrieve some relevant passage

50

00:10:29.860 --> 00:10:56.049

Jisun An: and then generate the answer, even using those retrieve the passages. Something. So the key idea here is rather than generating, based on the prompt they generate, based on a evidence of the documents that can be an evidence of that prompt. And this document that would be used as a evidence are coming from these retriever. So you assume that you have a set of

51

00:10:56.160 --> 00:10:58.390

Jisun An: proofful documents, and you are

52

00:10:58.580 --> 00:11:07.040

Jisun An: looking for retrieving those documents 1st and then using basic that generate the text. Given these passages

53

00:11:07.800 --> 00:11:27.709

Jisun An: so the normal flow would be so assuming that we have some kind of embedding model. The embedding model is something that we talked about in the all these stage of the our courses. So these are the model where, given a text, it encodes the text and transit to a any. Vector so these are the embedding models.

54

00:11:28.000 --> 00:11:38.649

Jisun An: So the the embedding model. Given any, input you can turn that into a vector so that they can be represented in the vector space. And then they can compute, based on these vector values

55

00:11:39.200 --> 00:12:05.680

Jisun An: to compare like different documents. So given that we have an embedding model given, we have some text that can be used as an evidence. If we are giving this text as an input to this embedding model, then they will be represented as our embedding, which is the factor, and assuming we have some database where stores all these factors, then these documents, vectors or text vectors will be saved into this vector database.

56

00:12:06.150 --> 00:12:19.319

Jisun An: And we are assuming that we have many different texture text that we can really rely on. So our vector database basically will contain many different documents which represented as a vector

57

00:12:19.590 --> 00:12:26.440

Jisun An: and then so this will be our like database that we constructed. Then given a query.

58

00:12:26.760 --> 00:12:54.009

Jisun An: the query is also the text, so we can use the same embedding model, and then query, will then this embedding model reserves in the query embedding. Then, using this query embedding, we search within this vector space. So if the query was here, then basically, we are looking at those other factors that is nearby to that query. So we are looking at the near list queries. So we assume that these are the documents that are similar enough to the document, so that it can help to answer to this particular query.

59

00:12:54.740 --> 00:13:13.709

Jisun An: So once we have these documents that are similar to our query, and assuming we also have separate generation model. We are you, then, now given this query, and given the retrieved documents, we are generating the some output. So this would be the general flow of the how the rag would work.

60

00:13:14.740 --> 00:13:29.000

Jisun An: I think idea itself is very simple, and at the same time has been quite effective in especially to tackling those issues that we just described. Any questions about this diagram? I think it's quite straightforward.

61

00:13:30.020 --> 00:13:35.639

Jisun An: But really like this is the essential thing to understand for the rage. So if you have any questions.

62

00:13:39.720 --> 00:13:40.470

Jisun An: yes.

63

00:13:42.170 --> 00:14:11.420

Jisun An: right? So the question is whether different embedding model may reverse indifferently. Yes, of course. So I will. Very soon. We'll talk about like how you can. I mean, we already talked about the different types of embedding models. So basically, you can use, like out of the box embedding models. But at the same time you can also train your own model to

64

00:14:11.600 --> 00:14:34.430

Jisun An: be better. So, I would say I mean larger models, larger embedding models will generally give better performance. But but then, even though that is larger, if your query is very domain specific, then it may not work well. So you may need to also fine tune the model to for your own data sets, and these are something that we will talk very soon. But thanks for the question, any other questions.

65

00:14:39.460 --> 00:14:40.170

Jisun An: Okay.

66

00:14:40.830 --> 00:15:10.086

Jisun An: all right. So, so most of the contents today will be about retrieving. So how can we? So the key parts of this rag system? Obviously the generation is also important. But as the our decoder model is getting larger and larger. I think generation, given the retrieved document part can be slightly easier. So most of the part that now people try to

67

00:15:10.890 --> 00:15:29.110

Jisun An: explore was this retriever part? So how can we then retrieve the documents? Given the query so there could be many different ways, and the easiest way that we can think is the sparse retriever is sparse. Retrieval, in other words, is like a keyword based retriever.

68

00:15:29.480 --> 00:15:39.260

Jisun An: So we express the query or the documents as a word, frequency, word, frequency. Vector, but usually it's normalized by the length.

69

00:15:39.360 --> 00:16:03.670

Jisun An: So, for example, if our query is, what is nlp, then we can represent this query as a set of I mean, vector, where each elements is representing each word. And and these values basically normalized by the length. So for Lot and Nlp and East, the 3 words have like 0 point 3 3. Right

70

00:16:04.330 --> 00:16:15.850

Jisun An: then, let's say that our documents is okay. So the 1st document was, What is life? Candy is life? So once again, these are having now 6, I mean, like 8

71

00:16:16.010 --> 00:16:35.050

Jisun An: tokens. And then you can basically count each of the words and then normalize it by the length. The second document is Nlp is an acronym for natural language processing. So you also have Nlp and ease has some value, and for the other words, it has zeros

72

00:16:35.410 --> 00:16:54.589

Jisun An: for the 3rd doll. I like to do good research on Nlp. So I mean, we omitted the rest of the words. But you can imagine, basically, it will have all the words across these documents and query, and then we have some value based on the existence of the words in that query or documents normalized by their length.

73

00:16:55.620 --> 00:17:18.399

Jisun An: So so this would be the easiest way to create or encode a text into the vector, because these are now each of the each of them are vector, so once we have a vector, we can basically find the document with the most similar documents or relevant query would be computed based on the inner product or the cosine similarity.

74

00:17:18.400 --> 00:17:34.249

Jisun An: So because these are the 2 vectors. So if you just compute for each of the document with the given this query, what is their inner products or the cosine similarity between the 2 vectors, then you can compute how relevant they are. So in this example in particular.

75

00:17:34.640 --> 00:17:58.930

Jisun An: basically for query and the documents, their inner product would be 0 point 1 6 5 for the document, 2 0 point 0 8 for document 3, it'll be 0 point 0 4. So if we are using this value as their relevance score, then this means that, given this query, d, 1 is the best results. But do you think this is the best results?

76

00:18:02.350 --> 00:18:14.829

Jisun An: So d 1 is, what is life? Handy is life. D. 2 is Nlp is an acronym for natural language processing. D. 3 is, I like to do good research on nlp, the query was, What is nlp.

77

00:18:15.280 --> 00:18:20.289

Jisun An: so which one? Which document is the better? One.

78

00:18:21.280 --> 00:18:25.430

Jisun An: 2 at least. Right? So I mean, obviously, I mean within within this world.

79

00:18:25.940 --> 00:18:42.989

Jisun An: So what we want is I mean the d 2 supposed to be, I mean looks better richers. But if you are using just a simple or the frequency based method, then these are basically not be able to find the good real relevance. And the I mean, one of the reason is

80

00:18:43.450 --> 00:19:05.760

Jisun An: here we assume all words are equally important, right? So we basically assume all words has equal weights. But as you know that some words may be more important. Some words may not be less important when you're computing this relevance, and that's the reason that you are. They introduced this Pfidf term weighting

81

00:19:06.110 --> 00:19:14.800

Jisun An: once again. These are like very basic Csir kind of information. But I think it is. I mean, it's anything's good to know. So

82

00:19:15.340 --> 00:19:30.379

Jisun An: so the idea some terms are more important than the others, and especially the idea is low frequency. Words are often more important. So, in other words, if you think about the articles like or ease these are free.

83

00:19:30.380 --> 00:19:55.700

Jisun An: frequent across all the documents. Right? So if you observe a word across all the documents, then basically, it means that it's not very important. But if you find a word in a very low number of documents, then maybe they should be considered to be more important. So that's the basic idea of the Tfidf, so the Tfidf is the short for time frequency and in document frequency.

84

00:19:56.080 --> 00:20:09.429

Jisun An: And their equation is defined in this way. So the term frequency is simple, like normalize the term frequency. So term frequency, divided by the total number of the sum of the old frequency.

85

00:20:09.900 --> 00:20:26.300

Jisun An: and the Idf. Is the term by. So the the d is the number of the entire documents divided by number of the documents that that particular term appeared. So if you think about if we have 100 documents.

86

00:20:26.300 --> 00:20:51.019

Jisun An: and if a term appeared in all 100 documents, then this term in the log will be one, because this 100 over 100, then log one will be 0. So basically it's worthless. But if there's if the if the words appeared only in one document, then this would be 100 divided by one. So the log 100 is larger value, so that will give some importance

87

00:20:51.020 --> 00:20:58.849

Jisun An: to the Tfidf metric. So by multiplying this tf and idf, this will give you the score for the

88

00:20:59.210 --> 00:21:07.859

Jisun An: for the and score, and then you can now represent the documents based on these values.

89

00:21:11.530 --> 00:21:27.770

Jisun An: but these are still like each word will have the Tfi depth score, and you will be able to represent the the documents based on similar to the sparse retriever, but rather than using, like the the pure value you just using the Tfi depth scores

90

00:21:29.179 --> 00:21:45.140

Jisun An: and then, as a variant of this tfidf, it's a Pm. 25, and this is one of the common methods, and also widely used, and apparently this has been beating many of the like trained models as well in utriever

91

00:21:45.750 --> 00:22:01.950

Jisun An: so it's similar to the Tf. Idf, but they just have a bit of smoothing into it. So the idf is the exactly the same, and for the tf, they just added a little bit of smoothing here. So so this part is if you are looking at

92

00:22:02.050 --> 00:22:22.020

Jisun An: these, the the red part they are basically adding this term to limit the the frequency itself. So it would be possible that the frequency of the words can be very, very large. So, but instead of they are normalizing, based, so the based on the the sum of the frequency. They just

93

00:22:22.020 --> 00:22:34.590

Jisun An: change it to this term, so that even though the frequency is very, very high, this value will be at maximum k plus one, because the the frequency term will be canceled, and you will only left at k 1 plus one.

94

00:22:35.420 --> 00:22:50.142

Jisun An: and then and then the the second part. So so they they are doing some kind of tom frequency saturation, so they limit. The value will be at maximum k 1 plus one. So even though the frequency is really really high, and then

95

00:22:50.610 --> 00:23:12.380

Jisun An: the second part is for the document length, normalization. So the Ld smarty here is the the length of the documents and the divided by the average length of the document. So if the average document length was 100, and if your document is like 5, then this value will be small, so the

96

00:23:13.190 --> 00:23:24.429

Jisun An: denominator is this more so, the the value will be large. In other words, if the word was, appeared in a document shorter document, then they consider it more important.

97

00:23:24.550 --> 00:23:37.659

Jisun An: and if I the word was, appeared in the longer document, then the importance of that word is basically decreasing. So that was the basic ideas of the DM 25. So both are

98

00:23:38.210 --> 00:23:50.882

Jisun An: a way to weight different terms Tfida once again, rather than just using the frequency. They also considered how often they occurred in the document.

99

00:23:51.610 --> 00:24:16.639

Jisun An: and so the idea is, basically, if the word is more frequently occurred across all the documents that is less important is not important, and if it is uniquely appeared, and it is more important. That was the idea of the Tfidf. And Bm. 25 is the variant of the Tfidf. And just adding these 2 aspects that they have this saturation for the Tom frequency.

100

00:24:17.020 --> 00:24:35.919

Jisun An: so that it will not be the value will not be based on the frequency value. And also they have this document length normalization. Now. So the the word, the term that appeared in a shorter document is more important than those words that appeared in the longer documents. So that's the Bm, 25.

101

00:24:36.880 --> 00:24:51.000

Jisun An: So once again, once you have this term weights, you can basically encode any of your queries or documents like this. But instead of these values, you will use Tfidf for the bm, 25 values.

102

00:24:51.720 --> 00:25:01.239

Jisun An: And similarly, so basically, you will still have 2 vectors, and you can use inner products or the cosine similarity to compare, to find which documents are most relevant to the query.

103

00:25:04.090 --> 00:25:31.299

Jisun An: And the 3rd one is the inverted, inverted index, and and these are just one of the technique for the database and you probably don't need to. I mean, usually no more. Database systems has already implemented this function. But here the idea is. So when you normally index any document you are indexing in this way. So for a given document, this document include this word, this word, this word.

104

00:25:31.300 --> 00:25:52.950

Jisun An: So d, 1 had what and candy and is, and d 2 has so and then inverted indexes. Now, if you are creating this kind of vector then. And then, if because across all the document, we have many, many tokens. So you will have many zeros in the. Vector so rather than you have index of vector, I mean, documented to the

105

00:25:53.300 --> 00:26:17.222

Jisun An: documented, to the words or the terms you are using inverted index. So instead of using the given the words like, what are the list of the documents? These words is included. So these are the inverted index. And basically, once you have this inverted index. Now you can retrieve what documents are containing this word. So these are like techniques that using in the database

106

00:26:18.100 --> 00:26:27.999

Jisun An: so just another thing that you can. So in this way you may be able to find, like what documents has this particular words.

107

00:26:28.550 --> 00:26:46.380

Jisun An: So in a very simple way, if you are looking for, I mean, if your query is, what is Nfp, then maybe you think that Nfp may be the most important term. So if you are using the inverted index. You can retrieve all the documents, including Nfp term. And then that could be maybe your relevance of documents.

108

00:26:47.950 --> 00:26:49.540

Jisun An: Any question up to here.

109

00:26:52.800 --> 00:27:04.459

Jisun An: Right? So so these are now or sparse retriever method. So what would be the issue? I mean? Obviously this one has the issue. So what would be the issue of this sparse retriever?

110

00:27:06.100 --> 00:27:09.829

Jisun An: So if you really want to find the most relevant documents

111

00:27:10.471 --> 00:27:17.329

Jisun An: and if you are using this parse retriever method, part of the any. What would be, I mean wouldn't work. Well.

112

00:27:22.010 --> 00:27:22.940

Jisun An: thank you

113

00:27:23.500 --> 00:27:51.069

Jisun An: right? Exactly. Exactly. So. They are good at finding the exact match terms. But basically they will not be able to consider any semantic relationship. So we've talked about this word representation. So there could be many different words that are having similar semantics. But then, basically, this sparse matrix is very hard. I mean, we'll not be able to capture any of them. And that's the reason where the dense retriever or embedding based retrievers comes in

114

00:27:52.260 --> 00:28:13.509

Jisun An: so the dense retriever is the we encode now documents or the query into the dense embeddings and find the nearest river. So basically, we embed the web encode this query into some embedding, and we also have multiple documents and encode into the dense embeddings. And basically, we are just finding

115

00:28:14.030 --> 00:28:39.490

Jisun An: comparing day 2 once again, because these are once again the vector so we still use the inner products or the cosine similarity. And then, now the question is, so, what kind of embedding should we can we use? So you can, we can use out of out of the box embedding. So those model that already have trained, or we can also use learned embedding. Or we can, we can train embeddings for a particular purpose.

116

00:28:40.880 --> 00:28:46.856

Jisun An: So so let's say, what embeddings can we use so

117

00:28:48.060 --> 00:28:52.290

Jisun An: what embedding can we use to encode a a sentence?

118

00:28:53.200 --> 00:28:57.140

Jisun An: Any any any model that in your head.

119

00:28:58.410 --> 00:29:05.220

Jisun An: But yes, of course we can use. And in the next slide any any other

120

00:29:05.400 --> 00:29:07.230

Jisun An: model can you think of?

121

00:29:12.000 --> 00:29:18.620

Jisun An: There was like world based model that we talked about in the word representation

122

00:29:21.230 --> 00:29:34.119

Jisun An: word to back. I don't know whether you remember that, but we also had this model word to back, which is the the model that encodes each word into the embeddings. Once again, I mean.

123

00:29:34.340 --> 00:29:35.150

Jisun An: it's.

124

00:29:35.280 --> 00:29:47.519

Jisun An: I assume, larger models definitely reduce in better performance, but still work. To make model sometimes is easy baseline, and it can beat also to a certain extent. So

125

00:29:47.650 --> 00:30:16.649

Jisun An: but let's think about word embedding a word to back model. So this model given, we have a sentences now. Each word can be represented to a embedding, and then so. But then we want a vector for this entire sentence. Right? So we need to somehow combine this and the easiest. And there could be different way to combine these vectors. But basically, we can just compute the average of these word embeddings. So because the word to back for a any word

126

00:30:17.103 --> 00:30:38.159

Jisun An: it reserves in the same size of the vector so you can simply get the average of those vectors that need to be some averaging that. Now this vector can represent this entire sentence. So so these are these are the way to create the Courier document embeddings, or it also called as a sentence embedding so

127

00:30:38.400 --> 00:30:48.720

Jisun An: sentence embedding in general is the way, is the model or the approach to encode a sentence to a like fixed size. Vector

128

00:30:50.090 --> 00:30:57.936

Jisun An: so this is the one way. And secondly, you can also use Bert Bert, or any other transformer based

129

00:30:58.870 --> 00:31:12.689

Jisun An: masked language models or the bidirectional transformer models which which we talked about, because these models are not necessarily good at generating, but they are good at encoding text, because they learned

130

00:31:12.690 --> 00:31:40.309

Jisun An: from both the directions, so they also the master language, if you recall masked language was, If you have a sentence, then they masked some words in within the sentences, and their goal was to predict these masked languages, using all the surrounding information. So these models, masked language models are usually not good at generating, because for generation you supposed to like mask, all the future tokens

131

00:31:41.480 --> 00:32:06.289

Jisun An: But this muscle language also use the information of the next kind of tokens, but but so they are not good at generating, but at the same time they are good at encoding the text, because they use both context. So we can use birds. And for the birds. If you are giving a sentence, or then each tokens will resulting it at the end of the layer. Each token will have some embedding. And now

132

00:32:06.290 --> 00:32:16.799

Jisun An: how can we? What out of these embeddings what can we use for to represent the sentence? One method we can just take this, this one token.

133

00:32:17.790 --> 00:32:36.770

Jisun An: and the reason that we can just use this last token, or the 1st token. We usually have the Cls token in front of the sentence, which I omit in this example, so we can either use the 1st Cls token or the last token. It doesn't matter, and we can use actually any of the token to represent a sentence.

134

00:32:37.050 --> 00:32:51.530

Jisun An: if we are using the bird, because when they compute the embeddings of each of the token, they already consider all the surrounding information so technically, this embedding already contains all the information, all the sentence.

135

00:32:51.820 --> 00:33:03.089

Jisun An: I I don't know whether there has been any experiment like which embedding which token embeddings are better at representing sentence. But technically, it is possible.

136

00:33:04.000 --> 00:33:33.409

Jisun An: But then, normally, the normally kind of use is basically we pull all the tokens and creating one token, basically, we can do average, or we can take maximum values of each of the vectors, or we can take minimum, may not be make sense. But there's 1 other thing which I don't remember at the moment. But there are like different pulling methods. And you can basically pull across all the tokens and create one embedding. And that will be your sentence embedding.

137

00:33:34.820 --> 00:33:40.060

Jisun An: does it make sense any questions?

138

00:33:47.040 --> 00:34:04.099

Jisun An: Okay, so the key is basically, there are different ways that, given a sentence the goal is create a vector, with the same same size, so that you can compare. So now we know that, like the Korean document, can be encoded. Using this method.

139

00:34:04.540 --> 00:34:33.950

Jisun An: and on these encoding kind of models there are a few remarks so normally bidirectional attention models like birds, or these variants, like Roberta or the modern birds. Any of these language models can be used to generate the sentence embeddings. So from this previous figure, basically, you can replace this bird model to something else, and larger models, or some variations are known to be better at doing the sentence embeddings.

140

00:34:34.780 --> 00:34:38.649

Jisun An: And now, people, because people know that the

141

00:34:39.020 --> 00:35:03.939

Jisun An: these days many large models are decode, only decoder only models. So there has. But then there are people. Are you? People need to encode this text. So there has been some efforts to change. I mean, transfer the this unidirectional attention models to the encoder models, and one of such effort is to add them to back. So they propose a method to given any decoder, only

142

00:35:03.940 --> 00:35:16.220

Jisun An: model to create an encoder model. But so the basic idea is, basically they took the decoder only model, and then they removed the masks and then train again, based on

143

00:35:16.290 --> 00:35:32.589

Jisun An: next next token, prediction on the masked words, and also they do. The next sentence, prediction as well. So they basically retrain these Rrms which are trained for as a decoder only model, borrowing the idea from the birds.

144

00:35:33.570 --> 00:36:01.139

Jisun An: But but if you are using this approach, you can transform any decoder, only model which are tend to be larger these days. And another interesting idea was the echo embeddings. So the limitation of the decoder only model in encoding text is simply because they never seen the next tokens, right? They only trained, based on the previous token. So the idea was very simple, so they repeat

145

00:36:01.140 --> 00:36:05.279

Jisun An: the sentences, so when they train

146

00:36:05.660 --> 00:36:29.380

Jisun An: so in that way. Now your previous, now your next token became your previous tokens. If you just concatenate the same 2 sentences back to back, and then you just train the model as it is then, even though this is the still unidirectional attention. But now you will also learn something from the future tokens. So that was the

147

00:36:29.480 --> 00:36:43.549

Jisun An: method. Codes are echoembedding. So there has been a couple of new approaches that are trying to transform the unidirectional attention to the bi-directional models so that it can be better at encoded models. So once again.

148

00:36:43.770 --> 00:36:46.389

Jisun An: encoder models are not

149

00:36:46.500 --> 00:36:55.350

Jisun An: for generation, but these are good for representing this. This text, which can be used in in different applications.

150

00:36:58.070 --> 00:37:05.930

Jisun An: So these were all about the dense retriever. And now let me talk about caveats of it and potential solutions.

151

00:37:06.430 --> 00:37:07.460

Jisun An: So

152

00:37:07.530 --> 00:37:32.349

Jisun An: so what would be? I mean? Obviously this dense retriever has been popularly used, but it is not without any limitations. And one of such limitation is basically we are simply computing the relevant score between query and documents, and even though the return reserves are not the exact answers, it still returns it right. So if you ask to find

153

00:37:32.350 --> 00:37:38.470

Jisun An: top 3 documents. Given this query, they will still return, even though these 3 are not exactly the answer.

154

00:37:38.480 --> 00:37:42.759

Jisun An: So what would be the potential solution that you can? You can tackle this

155

00:37:51.770 --> 00:38:19.430

Jisun An: well, simply assuming that there's a relevant score would be ranging from some like 0 to one. Then maybe you can set a threshold and how to determine this threshold. It will depend on your data set. So maybe you can check your data set. You may have a good set of query and the documents, and then you can see the distribution of those scores and see what is the reasonable threshold that you can think of. So basically, you can set a threshold to filter out the low confidence features.

156

00:38:20.330 --> 00:38:27.840

Jisun An: Another caveat would be. So answers may spend multiple sentences, but then, if you your

157

00:38:28.960 --> 00:38:49.109

Jisun An: if you have like sentence level I mean. So if you have, like the entire documents and try to match it, then maybe it would be too large to incorporate it when generated, outputs so, or it may not be the idea to be retrieved for the generation. So

158

00:38:50.160 --> 00:38:50.930

Jisun An: to.

159

00:38:53.100 --> 00:39:18.860

Jisun An: So there are different attempts. But they here, basically, they tried. One of the important issues in the rag was when you save all these documents into the vector database. So how should we chunk the documents or whether they should be chunk? Or if we are chunking, then how much, how? What like granularity that it need to be chunked. So that's the another consideration rather than the caveats or considerations to to think about.

160

00:39:20.245 --> 00:39:43.529

Jisun An: And then 3rd one is it's while the dense retriever is really good at finding semantically related documents, it actually struggle with finding exact phrases. So for some cases you may actually want to find the documents that has the exact keywords, but then sometimes it may fail because they are looking at the semantics.

161

00:39:44.350 --> 00:39:54.890

Jisun An: The solution would be basically rather than using dense retriever. Only you use basically the hybrid search, like both the semantic and the keyword base, and you combine the images

162

00:39:55.170 --> 00:40:15.640

Jisun An: and and lastly, models trained on one domain may not perform well in another domain. So I mean, even though the Wikipedia and all the entire Internet has a large data, and the Internet would really powerful on that. But that doesn't mean that it will work for the some other domain, especially like the legal text.

163

00:40:16.160 --> 00:40:25.710

Jisun An: So in this case solution would be, basically, you are now train your own model for encoding this domain specific data.

164

00:40:26.660 --> 00:40:34.529

Jisun An: So in the next couple of slides, I will talk about text chunking, hybrid search and training on the domain specific data.

165

00:40:35.050 --> 00:40:56.759

Jisun An: So chunking long text. I mean, I mean, yeah, not much to tell. But there has. I mean, there are like different way to chunk the long text. So you can basically chunk a sentence level or paragraph level. But then recently, they found that when so like, so like like the center level, would be too granular, the paragraph would be maybe good.

166

00:40:57.720 --> 00:41:24.189

Jisun An: But then the problem is, even the paragraphs also kind of is the within one document, so it may require some context before and after the paragraphs, so to do that they found that if you are overlapping windows of different paragraphs and and chunk in that way, then they found that this helps for the retriever. So if you are building your own vector database for the rag. This would be one consideration to think about

167

00:41:28.130 --> 00:41:29.476

Jisun An: and then

168

00:41:30.240 --> 00:41:56.259

Jisun An: the hybrid search. I mean, the idea is simple. So we have keyword based search. This will match exact words, and and precise, but it means, like the synonyms or the semantic meanings, semantic search. Like the dense retriever, they will good at understanding the meanings, but then they may retrieve rudely related. So people also have been using this hybrid search to combine both. Basically.

169

00:41:56.260 --> 00:42:07.360

Jisun An: So now you have given the same document even the query, you will have 2 different ranked list. One is based on the keyword, another is based on the semantic dense retriever.

170

00:42:07.610 --> 00:42:34.689

Jisun An: Then how can you combine these 2 and one way to combine these 2 is using the reciprocal rank fusion, and the idea is quite simple. It's the ensemble ranking method to combine 2 or multiple ranked lists of the search results. So given multiple ranked lists, it assigns its document, a score based on its rank in each of the list. Using this formula, I mean, it'd be easier to see the example.

171

00:42:35.140 --> 00:42:55.179

Jisun An: So, assuming we have 3 different documents we have. One is ranked by the Bm 25, and another based on the vector, based method embed the bird based sentence embeddings. Then. Now, the final result would be basically for each of the components you are using, one divided by this rank

172

00:42:55.540 --> 00:42:58.430

Jisun An: or yeah, one divided by one plus

173

00:42:58.540 --> 00:43:03.800

Jisun An: one plus rank so this may not be actually correct. One.

174

00:43:04.890 --> 00:43:27.870

Jisun An: Oh, no. The K is just the the small constants to prevent the yeah. So basically, I mean, simply speaking, it is one divided by the rank itself. So the the final results for the document a would be one divided one divided by one from the vm, 25 plus one divided by 3 from the vector based model. So the value will be 1.3

175

00:43:28.090 --> 00:43:43.100

Jisun An: for the document B, it would be 1.5 so, and the C would be 0 point 8 3. So by combining these 2, the results for the the B. Would be better than the A, so this would be some way to combine different ranked list.

176

00:43:43.920 --> 00:43:50.499

Jisun An: and I assume there could be many other method as well. But these are just one of popular methods that you can use

177

00:43:51.920 --> 00:43:53.050

Jisun An: any questions.

178

00:43:53.320 --> 00:43:54.490

Jisun An: Am I too fast?

179

00:43:55.770 --> 00:43:56.520

Jisun An: Okay?

180

00:43:57.170 --> 00:44:26.630

Jisun An: And thirdly, you can even once again. So even though you have a large model that encodes the text, well, still it may not be out of the domain, so you can still try to train your own model to learn retriever oriented embeddings. And so the basic idea and most commonly used idea is basically the constructive learning. So move the 2 positive documents more closer and the negative documents more further away.

181

00:44:27.592 --> 00:44:48.370

Jisun An: So, assuming that you have given a query or given the documents, you have positive or the negative documents you train using the contrastive laws. For example, Triplet loss is one of the popular one for these contrastive loss, which is basically the maximizing the distance between the

182

00:44:50.467 --> 00:45:05.410

Jisun An: no, it's it's actually, if the negative examples are greater than the positive documents, then it makes it 0. And so basically that that will not gonna happen. So that's the how the triplet loss is working.

183

00:45:06.824 --> 00:45:11.949

Jisun An: So. But the here the idea is basically you should have, is it?

184

00:45:12.450 --> 00:45:17.910

Jisun An: Oh, any questions about? Oh, I will give you some time to look at the Triplet loss

185

00:45:28.560 --> 00:45:49.160

Jisun An: so because it will take the Max value between the 2. If the distance between the positive example and so a is the anchor, P is the positive, and the N is the negative example. So if the anchor and the positives are, distance between them is greater than the one with the negative documents, then

186

00:45:49.530 --> 00:46:09.691

Jisun An: then this value will be the maximum higher than the 0, so they will use that as a loss, and then but if it's the other way around, if the negative value negative, the distance between the anchor, and the negative is higher, then it the 1st term will be the minus, so they will take the max, which will be the 0, so they will use this loss to

187

00:46:11.090 --> 00:46:13.850

Jisun An: updates their neural networks.

188

00:46:15.567 --> 00:46:17.420

Jisun An: And then so

189

00:46:17.580 --> 00:46:29.769

Jisun An: and there are different kind of laws to train these kind of examples but one of the key issue here, then, how can we get the negative examples?

190

00:46:30.620 --> 00:46:40.790

Jisun An: so one idea was the in batch negatives. So so, assuming that you created a data set of a pairs of the queries and associated documents.

191

00:46:40.790 --> 00:47:05.619

Jisun An: So now then, basically, you assume that that's the only the correct or positive answer. And all the other documents are just the negative examples. So if you have, like this data set where you have a set of query and documents. These documents are relevant to that query, then basically, that's the only the positive example. And you select the any other documents as a negative example. Then you can have this triplet of

192

00:47:05.830 --> 00:47:10.659

Jisun An: curry, positive documents and the negative documents, and you can train your model.

193

00:47:12.900 --> 00:47:36.490

Jisun An: But then there could be a problem in this approach I mean not problem. But this may not result in a good performance, because this may be just not enough hard examples. So the hard example is meaning that if we consider this other documents as a negative example, then they are likely to be like different topics. So, but what we want to do is even some documents really talks about the

194

00:47:36.490 --> 00:47:55.560

Jisun An: same topic. But that is not really relevant to the query. If we can use it as a negative example, then that will even improve the query. Search readers even better. Right? So even though this in batch negatives are one way and commonly used. But these usually doesn't include any hard negatives.

195

00:47:55.960 --> 00:48:09.619

Jisun An: So the way to create this hard negative would be so there could be in batch negatives. But you are using a weaker retriever, and to some other examples, and then treat them as a negative.

196

00:48:09.940 --> 00:48:23.209

Jisun An: So. But now, in this case these other documents are like ranked, based on the Bm. 25. So these documents are including probably any some of the terms that this query has.

197

00:48:23.310 --> 00:48:47.510

Jisun An: and then you can just consider them as a negative example. But the problem is, these negative might actually be a positive but still given still but then that there are chances that some of these regions are actually the positive, but we assume that that would be the rare cases, and we assume the the initial data set that we set the curry and the associated documents are the stronger

198

00:48:47.850 --> 00:49:01.099

Jisun An: positive example than any of these negatives. So that would be the way to pair Korean, positive and negative documents, and then you can use them to train the model to get the better encoding of these

199

00:49:01.640 --> 00:49:02.820

Jisun An: widgets.

200

00:49:03.890 --> 00:49:05.050

Jisun An: Any question.

201

00:49:14.830 --> 00:49:31.730

Jisun An: So so so what? So if you really want to improve the retriever accuracy. The in the previous in batch negative examples. They are more likely to be documents that are like, not relevant at all

202

00:49:32.040 --> 00:49:33.799

Jisun An: like different topics.

203

00:49:34.070 --> 00:49:47.559

Jisun An: So I was asking for what is Nfp. And the negative example was, what is sports or about sports. But then what we really want to know is, even though when so, if, when our curry was, what is Nfp, we could have different documents like

204

00:49:47.560 --> 00:50:08.680

Jisun An: Nlp. Is actually something something, or we would have like an essay about. I want to be a good researcher about Nfp, so within these 2, we know that the description of vector Nlp is the better and more relevant documents than the I love. Nfp, right? So these hard negatives are these documents, like

205

00:50:08.750 --> 00:50:20.170

Jisun An: documents, include some terms relating to the curry, but not exactly associate documents for the curry. So we want to distinguish these 2, and these are called as a hard, inevitable.

206

00:50:20.900 --> 00:50:30.920

Jisun An: So even though for the same topic there could be more relevant and less relevant documents. And those doc, those less document, less relevant documents, are the hard example of the hard narratives.

207

00:50:31.480 --> 00:50:38.119

Jisun An: So essentially, it's for improving the retriever performance or better, better features. Yeah.

208

00:50:42.130 --> 00:50:43.450

Jisun An: any other question.

209

00:50:47.140 --> 00:51:04.659

Jisun An: Alright. So as a reminder, we briefly talked about sentence, pert expert, and which is one of the most popular sentence embedding people have been using, which is really widely used. And so this is 1 1 such example that is learned

210

00:51:05.433 --> 00:51:12.476

Jisun An: I mean start starting from the word, and it is trained to encode the

211

00:51:14.020 --> 00:51:18.519

Jisun An: sentences. Better to find similar sentences.

212

00:51:19.378 --> 00:51:27.109

Jisun An: So the way that the sentence birth was trained. They use this by encoder model, where, if you have 2 sentences.

213

00:51:27.110 --> 00:51:39.080

Jisun An: 2 of them are, use the birth embedding, and then they are poor, then they are resulting in a 1 embeddings, as we discussed before, and then they simply compute it.

214

00:51:39.080 --> 00:52:03.499

Jisun An: Compare their cosine similarity. And these these are trained based on the data set where they have 2 sentences, and they actually had a label of how similar they are, so human were labeled on a pairs of sentences, and whether they are similar or not, and the expert was basically trained using this contrastive learning, there are 3 different ways to train the experts, and one of them was using the cosine similarity.

215

00:52:04.743 --> 00:52:06.239

Jisun An: To to train them

216

00:52:06.990 --> 00:52:27.519

Jisun An: so so in, if you are doing rag. Maybe you can start from the experts because it's already trained on trained to find the similar sentences better. But you can train your own data set on top of the experts using different laws, depending on what data set that you have.

217

00:52:27.610 --> 00:52:55.029

Jisun An: So we're not going to go through deeply. But but there are different ways to train the sentence embedding sentence. Translate, I mean the hugging face. Call them as a sentence transformer. So if you have pair of sentences and a label indicating how similar you are, then you can use like cosine similarity loss. If you have a pair of positive sentences without a label, you can use this multiple negative ranking loss. If you have a triplet, you can use triplet loss.

218

00:52:55.110 --> 00:53:01.640

Jisun An: So, depending on what kind of data set you have, you may be able to starting from like bird, I mean

219

00:53:01.730 --> 00:53:14.410

Jisun An: by encoder model, or like expert, which is already trained. You can train further using your own data set, and that will result in encoding your data better than the existing ones.

220

00:53:17.980 --> 00:53:18.890

Jisun An: Yeah.

221

00:53:19.160 --> 00:53:27.440

Jisun An: So the the next concept and we have this one and the evaluation, I think. And I think that would be all.

222

00:53:28.130 --> 00:53:33.790

Jisun An: One practical concept that's popularly used in the systems is the re-ranking

223

00:53:34.677 --> 00:54:02.570

Jisun An: so to discuss about the re-ranking we just talked about. But so think about how we can score the similarity between the 2 sentences. Then there will be 2 different ways. One is the by encoder, and the other is the cross encoder. So by encoder here means that basically we treat these 2 sentences independently, and then we encode them, using virtual or any other encoding encoders. And once we have

224

00:54:03.070 --> 00:54:18.640

Jisun An: the the output embeddings, then we simply compute their cosine similarity or the inner dot products. So that's the like by encoder model. So here, once again, we are treating the the query and the documents just independently

225

00:54:18.810 --> 00:54:47.921

Jisun An: the cross encoder model. We are actually jointly encode encode both query and C document. So now, when computing the I mean, so basically here they are not turning query documents into like the embeddings. But they just as an input, they get the query and the documents and the output. They will just provide the value. The similarity value from 0 to one. So basically, you can train the cross encoder models.

226

00:54:48.880 --> 00:55:01.270

Jisun An: to given the 2 sentences, what would be the similarity between the 2. So once again, the data set that we've been talking about for learning these models, you can also use the same data set to train the cross encoder models.

227

00:55:01.903 --> 00:55:20.049

Jisun An: So here in this page the most important thing that for you to understand the Bioencoder model, we are embed the Korean documents independently, and so each of these 2 will get their own embedding, which is showing here as a unv.

228

00:55:20.070 --> 00:55:38.050

Jisun An: and then we will. So these are the 2 vectors, and we will simply use, like cosine similarity or the inner dot products to compute their similarity in the cross encoder model. There's nothing like like transforming them into the embedding. But these are just a classifier so given. The query and the document, they will compute their similarity.

229

00:55:39.639 --> 00:56:08.270

Jisun An: Then, then, and the cross encoder. If we are looking at more details, then it would be something like this. So these are similar to like the transformer model that we've been seeing. So each the documents and the query will be separated into the tokens, and then they will compute the attentions between all these tokens to maximize to find their similarity score. So basically, you are training a model to output their similarity. So these are the cross encoder model.

230

00:56:09.020 --> 00:56:12.189

Jisun An: So now given these 2,

231

00:56:12.600 --> 00:56:28.739

Jisun An: So let's think about how much computation it would need. So given. We have 100 queries and 100 documents. How much compute is needed to identify similar sentences. So let's think about the by encoder first, st

232

00:56:29.160 --> 00:56:31.539

Jisun An: or or the crossing quarter first.st

233

00:56:32.800 --> 00:56:36.400

Jisun An: Either way is fine. So let's go with the by encoder. First.st

234

00:56:36.540 --> 00:56:43.490

Jisun An: So how many computation would we need to compute the

235

00:56:43.640 --> 00:56:51.299

Jisun An: similar document? So not the similar sentences similar document given each of the queries. So, how many computation would would it need?

236

00:56:55.980 --> 00:57:02.040

Jisun An: Maybe the question is not very clear here. But

237

00:57:03.100 --> 00:57:07.363

Jisun An: so let's think about one query. So for one query,

238

00:57:08.240 --> 00:57:37.930

Jisun An: in the by encoder example, the one query will be, just get their own embedding right. And then for the 1,000 document, we also need to transform this 1,000 document into 1,000 documents. Embeddings. Right? So the documents encoder will. All 1,000 documents will go through this document, encoder, and they will have 1,000 vectors, and you will now given this query, embedding and 1,000 document embedding you will find the most relevant documents.

239

00:57:39.040 --> 00:57:41.900

Jisun An: So now you can just repeat this 100 times

240

00:57:42.320 --> 00:57:48.029

Jisun An: to compute to get all the relevant documents for 100 queries.

241

00:57:49.580 --> 00:57:50.760

Jisun An: Does that make sense?

242

00:57:51.040 --> 00:58:05.509

Jisun An: So in terms of the computation here. What I meant was computing for computation I don't know, not. It's not like exactly embeddings, but but so it will require at least computing 100 plus

243

00:58:05.850 --> 00:58:11.179

Jisun An: 1,000 computation to get the embedding.

244

00:58:11.610 --> 00:58:27.659

Jisun An: because for the the 1st query it requires 1,000 documents for the second query. You can also use the same 1,000 documents embedding which you already computed in the previous prompt. So you don't need to re compute the

245

00:58:28.200 --> 00:58:32.489

Jisun An: document embedding for those 1,000 documents. Does that make sense?

246

00:58:38.940 --> 00:58:46.879

Jisun An: So for the documents? If you run 1,000 times for creating document embedding, you can use it for 100 queries.

247

00:58:47.180 --> 00:58:58.199

Jisun An: So basically, you will need 100 plus 1,000 computation to find the relevant documents for these 100 documents. So 1 1,100 computation is required.

248

00:58:58.910 --> 00:59:15.430

Jisun An: I think once again, this may not be the best question, but that is the how much computation they would would need. That's the best way that I can describe this. But now, if we talk about cross encoder, then for one query.

249

00:59:16.090 --> 00:59:20.850

Jisun An: you need to do matching, basically with 1,000

250

00:59:21.090 --> 00:59:31.199

Jisun An: documents and all 1,000 documents that need to go through this network because they need to find the best match between this query to order 1,000 documents.

251

00:59:31.980 --> 00:59:33.330

Jisun An: So in the 1st

252

00:59:33.490 --> 00:59:58.809

Jisun An: query, it also requires 1,000 plus one computation. But then for the second query, now, because they are computing attentions across the query and the documents. You cannot reuse the previous 1,000 documents unlike the by encoder. But for the cross encoder, basically, you need another 1,000 computation to find the similarity between the second query and all this 1,000 documents.

253

00:59:59.080 --> 01:00:00.819

Jisun An: So basically,

254

01:00:01.990 --> 01:00:12.829

Jisun An: in terms of the computation course, encoder will require 100 multiply by 1,000 times of the computation to get the similarity values, to find the most relevant documents

255

01:00:12.830 --> 01:00:41.950

Jisun An: so pros and cons by encoder definitely efficient, computationally efficient, but then the accuracy will be less accurate than the cross encoder, so cross, encoder, more expensive, but because we will look at the token level attentions, it will have a better resource in finding the relevance so cross encoder will be better at finding more relevant documents, but computationally expensive by encoder computationally more efficient. But then the accuracy may not be as good as the cross encoder.

256

01:00:43.430 --> 01:00:48.460

Jisun An: And so the idea of the re-ranking here is basically you have 2 steps.

257

01:00:48.590 --> 01:01:03.259

Jisun An: So in the 1st step, and they're assuming you have. And the reason that we are talking about re-ranking is in in most of the companies. They have in-house millions of documents to match to a particular query. So basically, you need

258

01:01:03.608 --> 01:01:23.839

Jisun An: you need a like efficient system, obviously. But then, even though the cross encoder is more accurate, you just cannot run this cross encoder across all the millions of the document. So what you do is in the 1st stage you are using something lighter, efficient, like keyword, based or semantic, based, or like something by encoder based.

259

01:01:24.260 --> 01:01:47.319

Jisun An: or even the hybrid one that we talked about which can be done very fast. So and then you get some initial search results like few 100 or 1,000. And then in the second stage, you re-rank this initial search results based on cross encoder based models so that it can find the better, more relevant documents. So that's the what is called as a re-ranking.

260

01:01:48.080 --> 01:01:49.349

Jisun An: So once again

261

01:01:49.370 --> 01:02:18.254

Jisun An: there could be many different way to score the similarity between the 2 sentences, and one is more cost, effective and efficient, but less accurate, another one is more expensive or more accurate. So basically, you are going through the 2 stage, 2 steps of retriever the relevant documents initially, and then you reramp using more accurate models. So that is the re-ranking. And it's been some standard system that has been used in the

262

01:02:18.870 --> 01:02:26.886

Jisun An: in the real world a few other comments so. And because now we talked about

263

01:02:27.560 --> 01:02:43.500

Jisun An: now, the these are probably more engineering issues, and I will briefly talk about. But now we have assuming we have like millions of documents. And we have a 1 query to match with so and the vector, database is something that I mean database for these vectors.

264

01:02:44.030 --> 01:02:45.489

Jisun An: And the issue is

265

01:02:46.590 --> 01:02:55.879

Jisun An: because you have millions of the data. If you are looking for like near list neighbor vectors. Then you will be just. You basically need to

266

01:02:56.030 --> 01:03:21.080

Jisun An: compute the similarity between query and all the 1 million documents. And then you need to select the top K right. That's the K near list, neighbor. Search right? But that's almost impossible. So would it be possible to retrieve these embeddings if sublinear time, because the linear is not impossible in practice. In every query. Linear time means that in every query you need to go through to match with your 1 million documents, which is not possible in practice.

267

01:03:21.080 --> 01:03:33.700

Jisun An: so can we do it in sublinear. And that's called as a method of the approximate near list, neighbor search. So even though we are not 100% sure that these are the documents that are the near list

268

01:03:34.320 --> 01:04:01.290

Jisun An: to the curry. But we just approximate it. So, and that's the techniques that have been used for the companies that are supporting for this vector database. And I don't have the slide here. But the normal, the traditional database companies. They haven't started to supporting this vector database. And there are also new companies that are supporting for the data vector database. And we will try one of them like in the in Thursday class as a lab.

269

01:04:02.230 --> 01:04:20.199

Jisun An: But anyhow, so there are 2 popular methods. One is the locality sensitive hashing, and this is actually, quite, very interesting idea. And so what they do is so assuming that we have different vectors here, and we just have one plane line, let's say this blue line.

270

01:04:20.990 --> 01:04:35.529

Jisun An: and we divide the space based on this line. So if the.is existing above this line, then we are giving one to this 1st digit, which is in in blue. And if the dots are below this line that we give 0.

271

01:04:36.570 --> 01:04:44.131

Jisun An: So we are hashing each of the vector to these values. And we are adding, let's say we have this

272

01:04:44.790 --> 01:05:00.028

Jisun An: red line dimension that we are splitting this space based on this line. And then the one that are on the left side of this red one. We also gave one to the second digit, and if it's the right we are giving the

273

01:05:00.680 --> 01:05:19.753

Jisun An: 0. And similarly, for those values on the for the green line, those dots on the right side of this green we give one and the left one we give 0. So now, these are like hashing techniques. So we know that which vectors are in which region and then we can use.

274

01:05:20.741 --> 01:05:28.549

Jisun An: Like inverted index to find. What are the documents that are? So once you have new, vector, then you can

275

01:05:28.550 --> 01:05:54.789

Jisun An: determine which region. This once you have a new query, then you can determine which region this query is likely, I mean will be in this space and given that hashing value like 1 0 1. Then you can. Now, using the inverted index, you can look for all the documents that, having this 101 as a value, and then from them you can do crossing encoder, or some other way to find the most relevant documents.

276

01:05:55.760 --> 01:06:02.320

Jisun An: And does it make sense kind of yes one.

277

01:06:02.480 --> 01:06:26.839

Jisun An: I don't think so. I I think there should be the way I mean you can I? I don't know exact methods, but there should be a way to find the line that are dividing the data in like most. So you can think of like equally. I mean, at least it should not be biased toward to a certain number. So if a line can divide the data into the half, I think that's the best better line.

278

01:06:26.840 --> 01:06:35.480

Jisun An: so there should be some way to find those rights. So it may not be random random but I don't know the exact methods today. But thanks thanks for

279

01:06:35.930 --> 01:06:37.527

Jisun An: any other question.

280

01:06:40.350 --> 01:07:03.139

Jisun An: Okay? And the next one is graph based search. So basically, you create some half so that you can go to the half first, st and then finding the the vectors nearby and given the time. I didn't add a slide here, but there's this method called as a hierarchical new.

281

01:07:05.340 --> 01:07:10.559

Jisun An: Maybe she will search so it's the HN.

282

01:07:13.460 --> 01:07:14.689

Jisun An: Something else

283

01:07:15.060 --> 01:07:33.019

Jisun An: I will. I will post it later. So that's 1 of the popular graph. Based. Search. but but the idea is basically you are firstly matching the half note that is most closer to you, and then you go one layer deeper, and then you kind of repeat that. So they will group some of the

284

01:07:33.487 --> 01:07:48.240

Jisun An: vectors, and then they just create a hierarchy so that you can. You can basically, it's like a tree search to do that. And these are the some of the method. And that's the implemented in different software like size and chrome from a Dv

285

01:07:49.222 --> 01:07:52.760

Jisun An: something that would be interesting in the rag.

286

01:07:54.620 --> 01:08:24.209

Jisun An: So so then, how? So? We have 2 more topics. So how can we evaluate the retriever? So the what is the good result? The relevant information should be in the top ranked, and it starts usually with some relevant judgment and try to match them. So let's say, we have highly relevant to some are relevant and the 0 relevance. And the I'm just building up for the finer metric. But let's say the 1st cumulative gain would be simply

287

01:08:24.569 --> 01:08:39.879

Jisun An: some of the relevant score at N value. Retrieve. So at N. Here is because when you are retrieving documents, you are usually retrieving top 3 or top 5. So these are the at end means the like. How many documents you want to retrieve.

288

01:08:40.720 --> 01:09:03.990

Jisun An: So the most common, easy way to compute this relevance would be so, for each of the ranked top 5 documents retrieved. You can measure whether that was relevant, somewhat relevant or relevant. Right? So let's say that you can give the score to each of these documents, then you simply just sum them up. So in this example the cumulative gain at 5 for ranking the score would be 5,

289

01:09:04.390 --> 01:09:23.580

Jisun An: and this one is the ideal ranking. So obviously, if you have the idea ranking, the most top one will be most relevant, these somewhat relevant and then not relevant at all. But then the problem here. So even the ideal ranking the sum will be 5. So these are just equal. So you cannot distinguish between the ranking and the idea ranking.

290

01:09:24.279 --> 01:09:48.909

Jisun An: So the cumulative gain will not work. So that's the reason the second one was the so you are now discount one divided by low 2, I plus one for lower ranked value. So basically, you are giving different scores for different ranking. So in this example, if we adopt this discount factor, then for the 1st value rather than you are summing the exact score. But you are basically

291

01:09:49.240 --> 01:10:01.459

Jisun An: do some penalty based on their ranking. So for the 1st ranked one, give one second 1 0 point 6 3 3, rd 1 0 point 5, etc. So if you are adding this discount, then the sum will be 0 point 6 2,

292

01:10:02.560 --> 01:10:14.800

Jisun An: and the idea ranking now will have 3.6 3. So basically, if you get the more relevant values in the higher ranking, then basically, it will be highly weighted. So you will have a higher score.

293

01:10:15.290 --> 01:10:39.279

Jisun An: But still this has also some other problem. So finally, they propose this normalized, discounted, cumulative gain. So they wanted to make sure that, like picking up good document, then it will lead to the good score. And basically they divided by this Dcg divided by the idea. Dcg, of the idea ranking. So that's the normalized, discounted, cumulative gain.

294

01:10:40.030 --> 01:10:47.309

Jisun An: So now we are looking at the DC. So how you can compute is simply here the results of the add 2.

295

01:10:47.770 --> 01:11:05.051

Jisun An: So if you look at these 2, Dcg will be 1.2 3. But that idea one will be 3.2 6, so the the N. Dcg. At 2 will be 0 point 3 8 6. So what this would actually mean is that at 2. These regions are,

296

01:11:05.410 --> 01:11:20.860

Jisun An: achieving the 38.6% of the idea results. So basically, these are the proportion compared to the idea ranking. So it can, the current ranking wizards. How good at replace recreating the

297

01:11:20.990 --> 01:11:35.650

Jisun An: the idea ranking. And if we are doing at 3, then also you can see that both are, I mean in the 3rd one they also had somewhat related document. So we see that this end value is actually increasing.

298

01:11:37.200 --> 01:12:01.700

Jisun An: So this is what this metric was intended to. But then, what if this 3rd value was not relevant? So if it was 0, then if you compute this, then this will be 0 point 3 4. So in compare with Ndcga 2 and Dcg, 3 would be actually lower. So they basically take consideration that if you pick up more good documents, then you, this score will get higher.

299

01:12:01.920 --> 01:12:07.619

Jisun An: So this is the Ndcg that we mostly commonly use to evaluate these wrenches.

300

01:12:09.270 --> 01:12:14.249

Jisun An: Sorry that I'm in a rush, and there are some other metrics like mean average precision.

301

01:12:14.630 --> 01:12:16.730

Jisun An: And this one is now

302

01:12:17.410 --> 01:12:31.729

Jisun An: they basically compute the precision value. But but if the if the documents is not relevant, then you don't take you don't just consider them so. Only those relevant documents. You compute the precision and get the average.

303

01:12:31.910 --> 01:12:51.830

Jisun An: And another metrics is the recall at one. So basically at top at top end, you compute the recall. So if there are 10 relevant documents for the query and at R at one will be 0, because there's no 0. Relevant documents. R at 2 will be one divided by one, because you retrieve only one over the

304

01:12:52.310 --> 01:12:57.149

Jisun An: and document and etc. So these are some other popular metrics that you can also use.

305

01:12:59.420 --> 01:13:11.448

Jisun An: The last thing is the retriever generator model, which I have 2 more slides. So so this was the actual paper that, firstly,

306

01:13:12.210 --> 01:13:14.740

Jisun An: introduce the term of the rag.

307

01:13:15.130 --> 01:13:25.910

Jisun An: I'm not going to go through the details of here. But here the the idea was so they realized that the the the aim was to generate the text

308

01:13:25.910 --> 01:13:48.059

Jisun An: given relevant documents that are retrieved. Given the query, so they create this end to end the system. Where, given a query, they retrieve the documents, and then they generate, based on this retrieve documents, and then, based on that reverse, they also trained both the generator and the retriever as well. So this was like the 1st

309

01:13:49.352 --> 01:14:04.723

Jisun An: the paper that that generate, I mean, like, introduce this idea of the rag. So both the retriever and the generator can be tuned to be better at doing this, so generating something based on the retrieve, retrieve documents on the current.

310

01:14:05.600 --> 01:14:26.600

Jisun An: Once again I will. I will not going to go through the details. But if you're interested in looking at the paper, or we can also talk. But then, as I told you initially how you will actually implement this rag will be. You can use some kind of retriever method to given a query out of all the document to get some text.

311

01:14:26.600 --> 01:14:35.699

Jisun An: and then you can use some kind of decoder model to generate. Given the query and using the retrieved document both as a prompt.

312

01:14:35.700 --> 01:15:01.700

Jisun An: So now that we have decoder model that have a larger context window. You can simply rather, you don't need to retrain the generator. But you can just simply use. I mean, add this retrieved documents as a part of their prompt, then the rest will be done. Based, I mean, the decoder model can take care of the rest of it. So that's the reason that we don't talk much about this last part of the generation.

313

01:15:02.660 --> 01:15:14.090

Jisun An: they also call this a grounded generation because they are grounded, based on the retrieved documents. But it seems that most of the time they are using they depends on the decoder model, because now they are very strong.

314

01:15:15.860 --> 01:15:17.160

Jisun An: So

315

01:15:18.270 --> 01:15:38.552

Jisun An: yeah, if you have all the costs you can use, like the Google to retrieve and then using the Gpt to generate the text given retrieve documents. So this would be the most simple implementation. And also we talked about different retriever models, and we talked about decoder models like previous lectures, so you can actually use them off the shelf.

316

01:15:39.180 --> 01:15:45.179

Jisun An: and then we will do some lab to introduce a little bit of the Rat on Thursday.

317

01:15:45.560 --> 01:15:49.550

Jisun An: I'm sorry that I'm already late. Any any questions?

318

01:15:52.820 --> 01:16:08.400

Jisun An: Alright! I hope this was quite straightforward so I will see you on Thursday, doing some exercise with the on on this particular topic. So yeah, have a great rest of the day, and I will see you on Thursday. Thank you.